DATC RDF-2021: Design Flow and Beyond

ICCAD Special Session Paper

Jianli Chen*, Iris Hui-Ru Jiang†, Jinwook Jung‡, Andrew B. Kahng§, Seungwon Kim§, Victor N. Kravets‡, Yih-Lang Li¶, Ravi Varadarajan§, and Mingyu Woo§

*Fudan University, Shanghai, China
†National Taiwan University, Taipei, Taiwan
‡IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
§UC San Diego, La Jolla, CA, USA
¶National Yang Ming Chiao Tung University, Hsinchu, Taiwan

***Abstract*—This paper describes the latest release of the DATC Robust Design Flow (RDF), *RDF-2021*, which has several key additions to expand its horizons. The Chisel/FIRRTL compiler is now part of DATC RDF, enabling support of recent hardware generator designs written in Chisel. Logic locking through RTL obfuscation, an updated ABC synthesis flow, and DFT support are other notable updates to the RDF. A Bookshelf-LEF/DEF converter powered by OpenDB is also added into DATC RDF's inventory as an enabler of robust benchmark conversion. We also describe efforts toward open metrics standards and datasets for machine learning (ML) applications and smart tuning of the design flow, as well as expansion of public analysis calibration data. Our paper closes with future research directions related to DATC's efforts.**

I. Introduction

IEEE CEDA Design Automation Technical Committee (DATC) [1] has developed a public reference design flow, named *DATC Robust Design Flow (RDF)*, over the past six years [2]–[7]. Its first release, named *OpenDesign Flow Database*, appeared in 2016 [2] and was built upon CAD contest-winning tools. The RDF subsequently evolved both *vertically* and *horizontally* to achieve a complete RTL-to-GDS flow with multiple tool options available [3]–[6]. In the 2020 release [7], RDF brought the integrated OpenROAD app [8] into its inventory, solidifying the RTL-to-GDS implementation flow. The RDF scope and mission were also updated, bringing attention to analysis and verification research.

In the past year, we have made several key updates that expand the horizons of the DATC RDF. We have included the Chisel/FIRRTL compiler [9] to support recent hardware generator designs written in Chisel. RTL obfuscation and DFT insertion are also added this year. Another noteworthy addition is an open-source Bookshelf-LEF/DEF conversion flow, *RosettaStone*, which brings past academic benchmarks and point tools to life with respect to usability in modern LEF/DEF-based full RTL-to-GDS flows. These additions improve the benefits brought by RDF as a research platform for a wide scope of hardware- and EDA flow-related research. New DATC initiatives this year also include definition of METRICS2.1, a common metrics system and exemplary datasets to support machine learning research, and development of a design flow autotuning framework. Calibration datasets to support research on analysis tools have also been expanded this year.

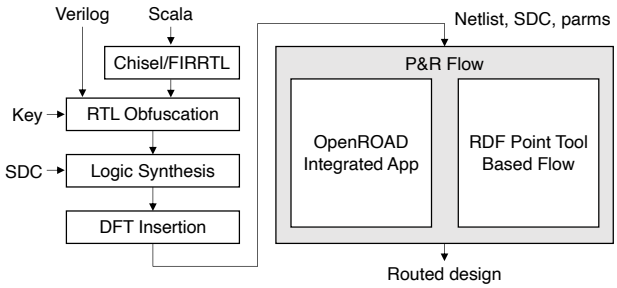

Fig. 1. RDF-2021 overview. The input design is Chisel-generated or pure Verilog RTL. The RTL can be locked using RTL obfuscation techniques. It is then synthesized and DFT-inserted, followed by P&R flow execution using the OpenROAD integrated app or the point tool-based RDF flow.

The rest of this paper is organized as follows. Section II gives an overview of the RDF-2021 flow, providing details of several key updates. Section III describes DATC's efforts toward a common metrics system, smart flow autotuning framework, and calibration dataset expansion. Future research directions and potential extensions are discussed in Section IV, and we provide some concluding remarks in Section V.

II. RDF-2021 Updates

Fig. 1 shows the RDF-2021 flow [10]. It starts from a pure Verilog RTL design or Chisel-generated Verilog. RTL can be optionally obfuscated using a recently-released RTL obfuscation tool [11]. Logic synthesis is then performed using Yosys with ABC, followed by DFT insertion with the Fault toolchain [12]. For P&R, RDF-2021 includes the OpenROAD integrated app [8] as well as the CAD contest point tool-based RDF flow [13]. Table I summarizes the supported components of DATC RDF-2021, where newly added and updated components are highlighted in boldface. In the rest of this section, we summarize the key updates of RDF-2021.

A. Chisel/FIRRTL and Hardware Generators

Chisel [9] is an open-source, domain-specific hardware design language embedded in Scala, consisting of a set of special classes, predefined objects, and language conventions for hardware design. Chisel aims to provide a modern platform for hardware design, whereby designers can utilize functional programming and object-oriented features of Scala. In Chisel, a hardware design is written in Scala and turned into Verilog via compilation and elaboration using the FIRRTL compiler.

TABLE I
RDF-2021 COMPONENTS

Component	Tools
RTL generator	**Chisel/FIRRTL**
RTL obfuscation	**ASSURE**
Logic synthesis	Yosys, **ABC**
DFT insertion	**Fault**
Floorplanning	TritonFP
Global placement	RePlAce, FZUplace, NTUPlace3, ComPLx, Eh?Placer, FastPlace3-GP, mPL5/6, Capo
Detailed placement	OpenDP, MCHL, FastPlace3-DP
Flip-flop clustering	Mean-shift, FlopTray
Clock tree synthesis	TritonCTS
Global routing	FastRoute4-lefdef, NCTUgr, CUGR
Detailed routing	TritonRoute, NCTUdr, DrCU
Layout finishing	KLayout, Magic
Gate sizing	Resizer, TritonSizer
Parasitic extraction	OpenRCX
STA	OpenSTA, iTimerC
Database	OpenDB
Libraries/PDK	NanGate45, SKY130, ASAP7, NCTUcell
Integrated app	OpenROAD
Benchmark conversion	**RosettaStone**

The intended usage of Chisel is to write "hardware generators" in an efficient and effective manner. This is the key differentiator of Chisel compared to traditional hardware description languages (HDLs) such as Verilog and VHDL. Using Chisel, it is possible to write highly-parameterized, abstract hardware generators using sophisticated programming techniques supported by Scala. The hardware generators can then be used to generate multiple implementations of different architectures. This allows hardware designers to explore various architectural design choices and best customize the resulting hardware implementations.

Chisel has garnered substantial interest from computer architecture and hardware design researchers, with open-sourced Chisel designs including Rocket Chip Generator [15], Gemmini deep-learning accelerator [16], and RISC-V Boom [17]. Enablement of Chisel in RDF-2021 opens the door to use of recent open-source processor and accelerator designs for design flow research. We also envision that RDF-2021 will benefit hardware design research by providing a readily usable, public design flow to evaluate multiple architectural and design choices.

B. RTL Obfuscation

Logic locking has emerged as an appealing technique to prevent piracy of hardware intellectual property (IP), and there have been rapidly evolving research efforts in this arena. In RDF-2021, we have included a recently-released RTL obfuscation-based logic locking technique, called ASSURE [11]. It is an RTL obfuscation framework assuming a netlist-only threat model where adversaries do not have access to activated chips. In ASSURE, an RTL design is obfuscated via three provably secure obfuscation techniques:

- **Constant obfuscation** eliminates selected constant values in RTL, and substitutes them with the secret locking key. The original RTL function is reproduced only when the correct key is provided.
- **Operation obfuscation** adds a redundant operator alongside an original operator, sharing the same inputs. A MUX is instantiated, which selects an output from the operators based on the provided locking key.
- **Branch obfuscation** obfuscates branch conditions in the original RTL so that the correct conditions are only activated when the correct locking key is provided.

With the RTL-based logic locking approach, ASSURE can hide essential design semantics effectively against hardware IP thefts. Usual gate-level locking techniques cannot protect such semantic information (e.g., constant values embedded in the original designs). Besides, as it directly obfuscates RTL, the ASSURE techniques are compatible with virtually any RTL-to-GDS flows.

```
abc 01> &cec -x cnn.170.aig cnn.187.aig
Networks are equivalent.  Time =  1238.07 sec
abc 02> &cec cnn.170.aig cnn.187.aig
Networks are equivalent.  Time = 12058.28 sec
```

Fig. 2. ABC `&cec` command runtime comparison with and without the new SAT solver. With the new solver, CEC runtime is greatly reduced.

C. ABC Synthesis Script Updates

The logic synthesis engine within RDF-2021 comes with improved scalability and new options to improve the quality of technology mapping. The scalability improvements are primarily due to the tight integration of a satisfiability solver with the internal design representation [18]. This eliminates creation of the auxiliary CNF representation of problem instances and instead performs reasoning directly in the underlying design model of a design. Such a tight coupling enables dramatic performance improvement of repetitive tasks as design size increases. For example, a verification task that checks combinational equivalence of two large neural networks exhibits $10\times$ runtime improvement as demonstrated in Fig. 2.

The performance improvement in the `&cec` command is due primarily to the new implementation of SAT sweeping capability that systematically examines the functional equivalence of signals. This is also used within the core logic optimizer `dch`, which combines *choice* computation [19] and SAT sweeping. The new implementation is packaged as `&dch -x` in the latest release and is the current default.

The released synthesis scripts in RDF are also updated to achieve better area and timing tradeoffs through a powerful optimization technique, dubbed *LazyMan synthesis* [20], that consults a database of logic rewriting choices empirically discovered in earlier runs. Such reuse of learnt data significantly reduces optimization runtime while enabling more accurate assessment of logic transformations. The recorded choices offer valuable know-how to the `if -y` algorithm [21] that explores sum-of-products logic realizations to improve timing.

D. DFT Support with Fault

In the open-source EDA ecosystem, there has not been a usable DFT solution, and the DATC RDF did not previously include DFT support, leaving it as a future direction [7]. Thanks to the recent release of the Fault tool chain [12], RDF-2021 now includes DFT support in its inventory. Fault

fills the previous gap by providing a complete DFT infrastructure including automatic test pattern generation (ATPG), scan chain connection, and JTAG interface insertion. The toolchain consists of five tools that enable ATPG and scan chain insertion, as follows.

- **ATPG.** A given netlist is first converted into a combinational network using `Cut`. Test vectors are then generated by `PGen`, subsequently compressed by `Compact`.
- **Scan insertion.** Given a netlist, test and shift enable signals are added by `Chain`, which also instantiates a scan MUX at each flip-flop input and creates scan chains. `Tap` creates a JTAG interface in the netlist.

In RDF-2021, the scan insertion toolchain of Fault is invoked after logic synthesis, as shown in Fig. 1.

E. Robust Conversion between Bookshelf and LEF/DEF

DATC RDF was initially built upon contest-winning academic tools. The flow was stitched with ad hoc format conversion scripts, as older academic tools are tightly coupled with specific contest input formats and do not have a shared/standard data model or format (e.g., LEF/DEF [22]). RDF-2019 [6] addressed this "interoperability" issue by integrating the OpenROAD tool chain that runs fully with industrial LEF/DEF format; RDF-2020 [7] further included the integrated OpenROAD app, which adopts an open-source physical design database, OpenDB [23], eradicating redundancies and inconsistencies between flow steps due to multiple separately-developed LEF/DEF parsers, as well as file- or name-based communication between the steps. Nevertheless, RDF's original flow based on academic point tools still depends on file-based LEF/DEF to Bookshelf conversions.

A previous effort toward horizontal extension of academic benchmarks for improved assessment of physical design research [24] proposes a horizontal benchmark extension infrastructure that allows commercial tools to accommodate existing CAD contest benchmarks, while realizing "apples-to-apples" assessment of commercial and academic physical design tools. However, the infrastructure shares the same limitation earlier versions of RDF, in that it depends on a script-based, ad hoc format conversion flow.

RDF-2021 includes an open-source robust benchmark conversion tool, *RosettaStone*, powered by OpenDB [23]. RosettaStone uses OpenDB as the central data model. It reads academic benchmarks written in Bookshelf, and creates OpenDB databases based on existing PDK information; missing information in the benchmarks, such as cell names and master cell types, are populated based on the given target PDK. The created OpenDB can be accessed directly via OpenDB APIs, or can be dumped into a DEF file. Fig. 3 shows Bookshelf-to-LEF/DEF conversion results of two ISPD-2005 placement contest benchmarks, obtained by using RosettaStone with the NanGate45 PDK. All the macro block sizes are properly scaled based on placement site definitions, and cell placement is performed using a commercial tool. RosettaStone is also capable of converting LEF/DEF into Bookshelf through OpenDB, enabling past academic tools to work within a modern industrial RTL-to-GDS flow for physical design research.

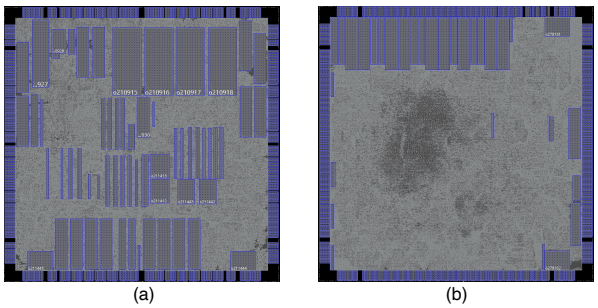

Fig. 3. Bookshelf to LEF/DEF conversion of the ISPD-2005 contest benchmarks using RosettaStone: (a) adaptec1 and (b) bigblue1. NanGate45 was used for this example. Standard cells were placed using a commercial tool.

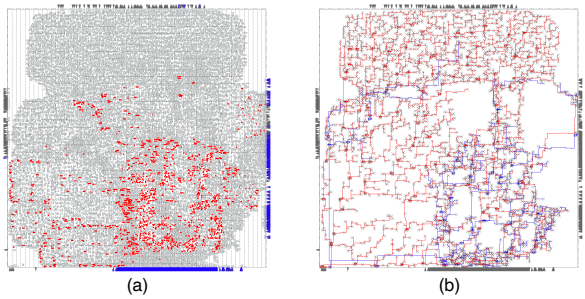

Fig. 4. P&R result for riscv-mini [25] obtained with the RDF-2021 flow. (a) Placement of obfuscation gates inserted by ASSURE (highlighted in red), and the locking key input ports (blue). (b) Test-enable (blue) and shift-enable (red) signals introduced by the Fault DFT toolchain.

F. Demonstration

We took a RISC-V 3-stage pipeline written in Chisel, riscv-mini [25], and ran through the RDF-2021 flow shown in Fig. 1. The Scala source files of riscv-mini were compiled into Verilog, which was then obfuscated using ASSURE with a 512-bit secret locking key and all the obfuscation techniques enabled. The obfuscated RTL was synthesized into a gate-level netlist targeting SKY130HD using Yosys with the updated ABC script. Fault was used to create scan chains in the netlist. Finally, we used the OpenROAD integrated app for P&R, obtaining a zero-DRC result from the TritonRoute detailed router. Fig. 4 shows the final routed design. A total of 1502 cells were introduced from ASSURE RTL obfuscation (total cell count is 25052). We also observe that the test and shift enable signals of scan chain, introduced by Fault, are properly connected and routed (see Fig. 4(b)).

III. Metrics, Smart Flow, and Calibration

Over the past year, the DATC has put new efforts toward establishing a common, standardized metrics system (METRICS2.1) and a smart EDA flow autotuning framework, as detailed in [26]. We have also expanded the calibration dataset, which was initiated in last year's DATC RDF updates [7].

A. Metrics

Research using academic and commercial tools has been actively conducted for decades throughout the RTL-to-GDS

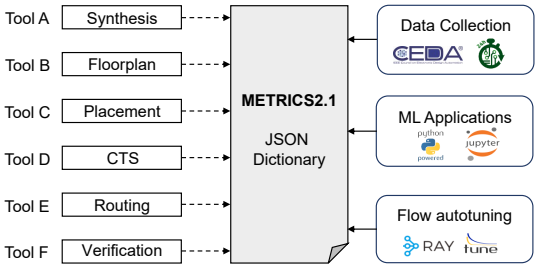

Fig. 5. Overview of METRICS2.1 infrastructure. METRICS2.1 captures design and tool parameter metrics from RDF-to-GDS flow execution, and stores the metrics data into a generalized/hierarchical JSON format. METRICS2.1 and collected metrics data can provide a common, standard foundation for ML CAD research.

domain, but has always suffered from a lack of standard reporting formats and tool metrics. Each tool or engine has its own, unique set of parameters that allow users to change optimization behaviors and thus trade off PPA and runtime metrics. The fragmentation of names and formats across the entire tool flow chain hampers sharing of machine learning models and know-how, as well as reproduction of results.

A METRICS1.0 infrastructure for the EDA and IC industries was proposed [27], [28] in the late 1990s to measure all design activity, mine all data, predict tool outcomes, find sweet spots or field of use for tools, and perform design-specific tuning of tools. In 2018, METRICS2.0 revisited the original goal of METRICS1.0, and proposed an updated architecture for collection and sharing of data for machine learning applications [29].

In this RDF update, we propose METRICS2.1 as a new standard for metrics collection and design process recording [26]. The goals of METRICS2.1 are (i) to provide a standardized naming and format for design tool and flow metrics data, and (ii) to define a robust structure for large-scale metrics archives. As shown in Fig. 5, METRICS2.1 provides integrated metrics extracted from the complete RTL-to-GDS flow, from synthesis to detailed routing to verification. We also provide thousands of RTL-to-GDS metrics datasets along with all configuration files needed for complete reproducibility, in the DATC's GitHub [30]. Example data analyses and ML applications in the form of Jupyter notebooks are also shared as a guide for future research with large metrics datasets.

B. Design Flow Autotuning

The complete RTL-to-GDS flow generally includes a variety of tool options and "recipes" at each flow stage. In practice, choosing such tool options and recipes to obtain the best possible PPA from EDA tools requires expert intuition and experience of designers. With this backdrop, we posed a "smart RDF flow" challenge in [7], and have added to RDF-2021 a design flow autotuning framework, named *AutoTuner* [26]. The AutoTuner framework provides "no-human-in-loop" parameter tuning for commercial and academic RTL-to-GDS flows; it is available as open source under the DATC GitHub organization [31].

Fig. 6 shows an overview of the proposed AutoTuner framework. Inputs include parameter configuration and reward

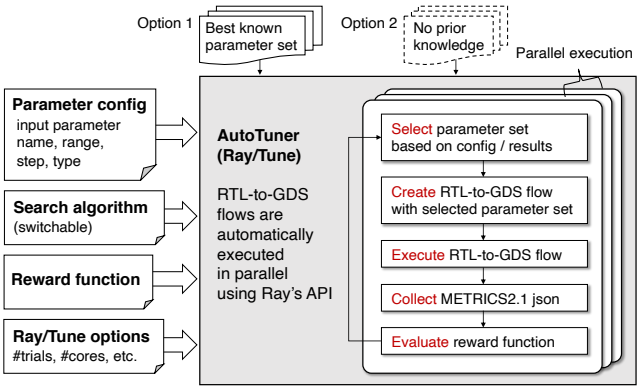

Fig. 6. Flow autotuning framework with inputs that include parameter configuration, search algorithm, reward function definition, and job execution option. The tuning can start from best-known parameters, or with no prior knowledge.

TABLE II
FLOW AUTOTUNING RESULT WITH VARIOUS SEARCH ALGORITHMS

Algorithm	Eff. clock period (ns)	Utilization (%)	Power (mW)
Baseline	17.78	20	52.3
PBT [33]	15.16 (−15%)	37 (46%)	14.3 (−73%)
HyperOpt [34]	14.94 (−16%)	40 (50%)	28.6 (−45%)
Ax [35]	16.04 (−10%)	26 (23%)	17.2 (−67%)
Optuna [36]	14.87 (−16%)	42 (52%)	27.8 (−47%)
Nevergrad [37]	15.23 (−14%)	18 (−11%)	19.4 (−63%)

function, as well as a reference EDA flow script. A parameter configuration is defined as a generic JSON object, enabling the AutoTuner to easily support various tools and flows. The AutoTuner framework uses METRICS2.1 to capture PPA of individual search trials. Users can explore various reward functions that steer the flow autotuning to different PPA goals. The proposed framework further supports various parameter search algorithms, e.g., [33]–[37]. Users can select a search algorithm based on their preference, or try multiple search algorithms at the same time and pick the best result.

To illustrate use of the AutoTuner framework, we show parameter tuning of the OpenROAD tool flow with SKY130HD using various search algorithms. The testcase is a public RISC-V based design, *ibex_core* [32], with baseline design result obtained using a manually-tuned parameter set. We pick 11 design and tool parameters, and define the reward function as a weighted sum of total power, effective clock period (i.e., target clock period minus worst setup timing slack), and total cell area. For fair comparison, all parameter tuning jobs are given the same wall-time constraint (17 hours), rather than the same total number of trials budget; this is because each search algorithm has a different tradeoff of exploration and exploitation. Table II shows the parameter tuning results obtained with various search algorithms [33]–[37]. We observe that most of the search algorithms are able to improve PPA of the design outcome; for timing and area, Optuna gives the best result, while PBT achieves the lowest power.

We note that the superiority of search algorithms may vary by design, parameter sets, reward function, etc. Hence, it is important to support multiple search algorithms, with a flexible interface to easily define parameter sets and reward functions. The benefit of this capability in AutoTuner has been confirmed

in multiple contexts such as that described above. We thus envision that AutoTuner can serve as a general parameter tuning framework for various RTL-to-GDS flows.

C. Calibration

As described in [7], RDF includes a calibrations repository to support academic research in electrical analysis and other design verification tasks. This repository provides a collection of reference analysis and verification results in open-source enablements. Over the past year, the collection of calibrations has expanded through contributions by POSTECH, the University of Minnesota, IBM Research, and UC San Diego. The new calibrations data are derived from runs of the latest OpenROAD flow with three PDKs (NanGate45, SKY130HD, SKY130HS) and four designs (gcd, aes_cipher_top, jpeg_encoder, ibex_core), using three different floorplan utilization and placement density configurations. Furthermore, we have introduced new scripts to introduce *noise* in the open-sourced calibration data, to enable obfuscations. New calibration data includes SPEF, 5-worst and endpoint slack, and IR drop JSON files with anonymization through this noise addition. The underlying anonymization scripts are available in the repository [39].

Our noise introduction method defines a *relative*, i.e., multiplicative, perturbation noise n as a random variable drawn from a normal distribution $N(1, \sigma_n)$, where σ_n is a predefined target standard deviation of noise. Thus, for example, around 95% of the noise perturbations will fall within $[1 - 2\sigma_n, 1 + 2\sigma_n]$. A maximum allowed noise $n_{\max}$ prevents too-large perturbations of original data; if the sampled value of n exceeds $n_{\max}$ for a given noise perturbation, we regenerate the perturbation until the constraint is satisfied. The values of σ_n and $n_{\max}$ are set differently for PEX, STA, and IR drop analyses. We summarize our noise introduction method as follows.

- For PEX, we apply the open-source SPEF parser [38], then multiply R and C values on each segment by n, and update the total capacitance of each net accordingly.
- For timing, AAT and RAT are multiplied by n at each pin in the timing graph. We then update the AAT and RAT of timing endpoints accordingly.
- For IR drop, voltage values are multiplied by n.

For timing noise, we observe empirically that the accumulated endpoint AATs generally have only a few ps of noise if we set σ_n and $n_{\max}$ as 0.01 and 0.02, respectively. Thus, the noise introduction does not appear to compromise the research value of the calibration data.

IV. Future Directions

In this section, we outline potential research directions that can take advantage of RDF-2021. We also discuss possible extensions that are of particular interest for RDF's future.

A. Intelligent Logic Synthesis

In the currently-released flow, the high-level components of an input Verilog are converted into the interconnection of bit-level primitives before the invocation of ABC logic optimization. Such an early expansion leads to losing a design's high-level intent, subsequently restricting the domain of logic optimizations to a bit-blasted netlist. A front-end availability of a complete RTL capture of the Verilog would enable ABC to integrate more "intelligence" into its reasoning. For example, interpreting the arrays and memory components at the logic reasoning level would open new algorithmic opportunities to improve routing congestion and area-timing tradeoffs.

B. Logic Synthesis Acceleration Techniques

The logic synthesis methods in the currently-released flow successfully adopt the SAT engine to repeatedly solve problem instances cast in the propositional form. However, to further increase the optimization returns, more expressive logic than SAT is needed. The formulations that reason over first-order logic with some additional theoretic constraints (e.g., bit-vector arithmetic) – known as Satisfiability Modulo Theories (SMT) – have exhibited significant success in recent years. Their evolving engines broaden applications within the design automation flow. The currently deployed front-end driver SymbiYosys [40] for Yosys-based formal verification hosts a subset of open-source SMT solvers [41]–[43], and serves a starting point for exploration in logic synthesis.

Furthermore, the recent release of the ABC logic synthesis engine enables substantial runtime reductions through the deep integration of the SAT solver with its core optimization tasks. However, the popularization of large machine learning accelerators continues to stress scalability demands on logic optimization. The broader integration of GPUs into the fine-grained transformations (e.g., "rewriting" and technology mapping) is one exploratory avenue to further shorten turnaround times. Similarly, multi-core processors can help parallelize compute-intensive synthesis tasks to analyze large portions of a design simultaneously.

C. Modern and Complete Benchmarks for Physical Design

Session 4 of ISPD-2021, "Driving Research in Placement: a Retrospective," conducted a poll on advancing placement research [45]. One of the questions was on the topic of placement benchmarks. All 39 respondents agreed that more benchmarks are needed to advance placement research; 54% in particular pointed out that "*realistic benchmarks reflecting advanced technology nodes*" are essential, and another 38% emphasized the necessity of a complete benchmark suite including timing and power characterization. Kahng [44] also points out the urgency of more complete and modern benchmarks to drive research on CAD optimizations toward next-generation capability.

To this end, it is important to enable future-looking research with (i) realistic, advanced standard-cell libraries as well as (ii) modern designs. For (i), PROBE2.0 [46] provides a framework to generate realistic cell libraries and collaterals (tech/cell LEFs, Liberty, etc.) with different technology parameters, as well as a platform for assessing the quality of the generated libraries. Notably, PROBE2.0 can be used to provide realistic

standard-cell libraries that reflect advanced technology parameters. For (ii), Chisel-based, advanced hardware generators can also help to provide modern designs, e.g., RISC-V cores [15] and AI accelerators [16], which are readily usable with full functionality for synthesis and physical design research.

D. Extensive Design and Flow Co-Tuning

The scope of the flow autotuning can be further extended to cover a more complete design flow and process. As one example, the ASSURE RTL obfuscation [11] performs various RTL obfuscations to secure a given design; each obfuscation technique has different impacts on PPA and security. We may include the RTL obfuscation stage in the flow tuning to obtain the best design outcome for PPA as well as security. Hardware design with Chisel can also benefit from extension of the flow autotuning to explore both the architectural design and flow parameter solution spaces, making it possible to co-optimize the design and implementation flow.

V. Conclusion

In this paper, we have described the latest release of the IEEE CEDA DATC Robust Design Flow. With RDF-2021, we can now take advantage of recent hardware generator designs written in Chisel, ranging from RISC-V processors to deep-learning accelerators, to assess and improve the academic RTL-to-GDS flow using advanced hardware designs. Logic locking with RTL obfuscation is another key addition to the flow, which makes security-related research possible with RDF-2021. DFT support with the recently-released open-source Fault tool makes the flow even more complete, and the RosettaStone framework built on OpenDB enables a robust Bookshelf-LEF/DEF conversion with well-defined underlying data model. All of these updates expand DATC RDF's horizons towards a more complete platform for hardware design and flow research. We also present new initiatives toward common metrics standardization to support AI/ML applications, along with a flow autotuning framework having a generic and flexible interface to support various design flows and search algorithms. The calibration dataset has also been expanded, with an open-sourced noise introduction script. Finally, we have noted several research challenges that can take advantage of RDF-2021 and are related to the DATC RDF's future.

VI. Acknowledgments

This work was supported by IEEE Council on Electronic Design Automation (CEDA). We thank Schuyler Eldridge, Christian Pilato, Alan Mishchenko and Mohamed Shalan for supporting tool integrations under RDF-2021 and for valuable discussions.

References

[1] IEEE CEDA Design Automation Technical Committee. https://ieee-ceda.org/node/2591.
[2] J. Jung, I. H.-R. Jiang, G.-J. Nam, V. N. Kravets, L. Behjat, and Y.-L. Li, "OpenDesign Flow Database: The infrastructure for VLSI design and design automation research," *Proc. ICCAD*, 2016.
[3] J. Jung, P.-Y. Lee, Y. Wu, N. K. Darav, I. H.-R. Jiang, V. N. Kravets, L. Behjat, Y.-L. Li, and G.-J. Nam, "DATC RDF: Robust design flow database," *Proc. ICCAD*, 2017.
[4] J. Jung, I. H.-R. Jiang, J. Chen, S.-T. Lin, Y.-L. Li, V. N. Kravets, and G.-J. Nam, "DATC RDF: An academic flow from logic synthesis to detailed routing," *Proc. ICCAD*, 2018.
[5] —, "DATC RDF: An open design flow from logic synthesis to detailed routing," *Proc. Workshop on Open-Source EDA Technology*, 2018.
[6] J. Chen, I. H.-R. Jiang, J. Jung, A. B. Kahng, V. N. Kravets, Y.-L. Li, S.-T. Lin and M. Woo, "DATC RDF-2019: Towards a complete academic reference design flow," *Proc. ICCAD*, 2019.
[7] —, "DATC RDF-2020: Strengthening the foundation for academic research in IC physical design," *Proc. ICCAD*, 2020.
[8] The OpenROAD Project. https://github.com/The-OpenROAD-Project
[9] Chisel/FIRRTL. https://www.chisel-lang.org
[10] DATC Robust Design Flow. https://github.com/ieee-ceda-datc/datc-rdf
[11] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg and R. Karri, "ASSURE: RTL locking against an untrusted foundry," *IEEE TVLSI*, 2021.
[12] M. Abdelatty, M. Gaber and M. Shalan, "Fault: Open-source EDA's missing DFT toolchain," *IEEE Design & Test*, 2021.
[13] DATC RDF-2020. https://github.com/ieee-ceda-datc/RDF-2020.git
[14] ABC: System for Sequential Logic Synthesis and Formal Verification. https://github.com/berkeley-abc/abc
[15] Rocket Chip Generator. https://github.com/hrsatgithub/rocket-chip
[16] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," *Proc. DAC*, 2021.
[17] The Berkeley Out-of-Order RISC-V Processor. https://boom-core.org
[18] H.-T. Zhang, J.-H. R. Jiang, L. Amaru, A. Mishchenko, and R. Brayton, "Deep integration of circuit simulator and SAT solver," *Proc. DAC*, 2021.
[19] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," *Proc. FPGA*, 2010.
[20] W. Yang, L. Wang, and A. Mishchenko, "Lazy man's logic synthesis," *Proc. ICCAD*, 2012.
[21] A. Mishchenko, R. Brayton, S. Jang, and V. N. Kravets, "Delay optimization using SOP balancing," *Proc. ICCAD*, 2011.
[22] LEF/DEF Reference 5.8. https://si2.org/oa-tools-utils-libs/
[23] T. Spyrou, "OpenDB, OpenROAD's database," *Proc. Workshop on Open-Source EDA Technology*, 2019.
[24] A. B. Kahng, H. Lee and J. Li, "Horizontal benchmark extension for improved assessment of physical CAD research," *Proc. GLSVLSI*, 2014.
[25] riscv-mini. https://github.com/ucb-bar/riscv-mini
[26] J. Jung, A. B. Kahng, S. Kim, and R. Varadarajan, "METRICS2.1 and flow tuning in the IEEE CEDA Robust Design Flow and OpenROAD," *Proc. ICCAD*, 2021.
[27] S. Fenstermaker, D. George, A. B. Kahng, S. Mantik, and B. Thielges, "METRICS: A system architecture for design process optimization," *Proc. DAC*, 2000.
[28] A. B. Kahng and S. Mantik, "A system for automatic recording and prediction of design quality metrics," *Proc. ISQED*, 2001.
[29] S. Hashemi, C. T. Ho, A. B. Kahng, H. Y. Liu and S. Reda, "METRICS 2.0: A machine-learning based optimization system for IC design," *Proc. Workshop on Open-Source EDA Technology*, 2018.
[30] Metrics4ML. https://github.com/ieee-ceda-datc/datc-rdf-Metrics4ML
[31] DATC RDF AutoTuner. https://github.com/ieee-ceda-datc/datc-rdf-flow-tuner
[32] Ibex RISC-V Core. https://github.com/lowRISC/ibex.
[33] M. Jaderberg *et al.*, "Population based training of neural networks," *arXiv preprint 1711.09846*, 2017.
[34] Hyperopt: Distributed Hyperparameter Optimization. https://github.com/hyperopt/hyperopt
[35] Ax–Adaptive Experimentation Platform. https://ax.dev/
[36] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *Proc. KDD*, 2019.
[37] Nevergrad. https://gitHub.com/FacebookResearch/Nevergrad
[38] Parser-SPEF. https://github.com/OpenTimer/Parser-SPEF
[39] DATC RDF Calibrations. https://github.com/ieee-ceda-datc/datc-rdf-calibrations
[40] SymbiYosys. https://github.com/YosysHQ/SymbiYosys
[41] The Yices SMT Solver. http://yices.csl.sri.com
[42] Z3 Theorem Prover. https://github.com/Z3Prover/z3
[43] Boolector. https://github.com/boolector/boolector
[44] A. B. Kahng, "Advancing placement," *Proc. ISPD*, 2021.
[45] I. Bustany, J. Jung, P. H. Madden, N. Viswanathan, and S. Yang, "Still benchmarking after all these years," *Proc. ISPD*, 2021.
[46] C.-K. Cheng, A. B. Kahng, H. Kim, M. Kim, D. Lee, D. Park and M. Woo, "PROBE2.0: A systematic framework for routability assessment from technology to design in advanced nodes," *IEEE TCAD*, 2021.